INTRODUCTION TO TRANSACTION PROCESSING

CHAPTER 21 (6/E)

CHAPTER 17 (5/E)

WHY WE NEED TRANSACTIONS

- A database is a shared resource accessed by many users and processes concurrently
- Not managing this concurrent access to a shared resource will cause problems (not unlike in operating systems)
  - Problems due to concurrency
  - Problems due to failures

LECTURE OUTLINE

- Basic Terminology
- Desirable Properties of Transactions
- Transaction Support in SQL

DEFINITIONS

- **Transaction**: an application-specified, *atomic* and *durable* unit of work (a process) that includes one or more DB access operations
  - Example from banking database: Transfer of $100 from a checking account to a savings account
  - Characteristic operations
    - **Reads** (database retrieval, such as SQL SELECT)
    - **Writes** (modify database, such as SQL INSERT, UPDATE, DELETE)

- **Online Transaction Processing (OLTP) Systems**: Large multi-user database systems supporting thousands of concurrent transactions (user processes) per minute

SOME TERMINOLOGY

- Transaction boundaries: *Begin_transaction* and *End_transaction*
- Transaction may be:
  - *Stand-alone*, specified in a high level language like SQL submitted interactively, or
  - More typically, *embedded* within application program
    - Application program may include specification of several transactions separated by Begin and End transaction boundaries
    - Transaction code can be executed several times (in a loop), spawning multiple transactions
- Transactions can end in two states:
  - Commit: transaction completes successfully and its results are committed (made permanent)
  - Abort: transaction does not complete and none of its actions are reflected in the database

REMEMBER (WHY WE NEED TRANSACTIONS)

- A database is a shared resource accessed by many users and processes concurrently
- Not managing this concurrent access to a shared resource will cause problems (not unlike in operating systems)
  - Problems due to concurrency
  - Problems due to failures

TRANSACTION PROCESSING MODEL

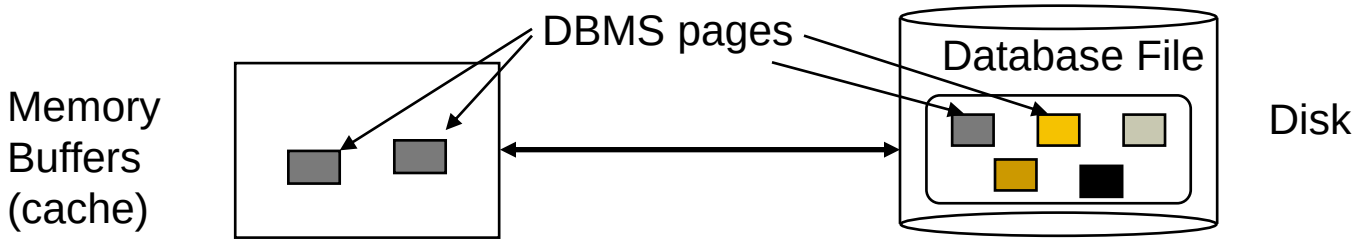
- Simple database model:
  - Database: collection of named data items
  - **Granularity** (size) of each data item immaterial
    - A field (data item value), a tuple, or a disk block
    - Transaction processing concepts are independent of granularity

- Basic operations on an item *X*
  - **read_item(*X*)**: Reads a database item *X* into a program variable
    - For simplicity, assume that the program variable is also named *X*
  - **write_item(*X*)**: Writes the value of program variable *X* into the database item named *X*

- Read and write operations take some amount of time to execute

READ AND WRITE OPERATIONS

- Basic unit of data transfer from the disk to the computer main memory is one disk block (or page)



- **read_item(*X*)** includes the following steps:
  1. Find the address of the disk block that contains item *X*
  2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer)
  3. Copy item *X* from the buffer to the program variable named *X*
- **write_item(*X*)** includes the following steps:
  1. Find the address of the disk block that contains item *X*
  2. Copy that disk block into a buffer in main memory (if it is not already in some main memory buffer)
  3. Copy item *X* from the program variable named *X* into its correct location in the buffer.
  4. Store the updated block from the buffer back to disk
     - either immediately or, more typically, at some later point in time

WHAT CAN GO WRONG?

- Consider two concurrently executing transactions:

	at ATM window #1
1	read_item(savings);
2	savings = savings - $100;
3	write_item(savings);
4	read_item(checking);
5	checking = checking + $100;
6	write_item(checking);

	at ATM window #2
a	read_item(checking);
b	checking = checking - $20;
c	write_item(checking);
d	dispense $20 to customer;

- Systems might crash after transaction begins and before it ends
  - Money lost if between 3 and 6 or between c and d
  - Updates lost if write to disk not performed before crash
- Checking account might have incorrect amount recorded:
  - $20 withdrawal might be lost if T2 executed between 4 and 6
  - $100 deposit might be lost if T1 executed between a and c
    - In fact, same problem if just 6 executed between a and c

ACID PROPERTIES

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all
- **Consistency preservation**: A correct execution of the transaction must take the database from one consistent state to another
- **Isolation**: Even though transactions are executing concurrently, they should appear to be executed in isolation; that is, their final effect should be as if each transaction was executed alone from start to end.
- **Durability**: Once a transaction is committed, its changes (writes) applied to the database must never be lost due to of subsequent failure

- Enforcement of ACID properties by a DBMS:
  - Database constraint subsystem (and application program correctness) responsible for *C*
  - Concurrency control subsystem responsible for *I*
  - Recovery subsystem responsible for *A* and *D*

TRANSACTION SUPPORT IN SQL

- A single SQL statement is always considered to be atomic
  - Either the statement completes execution without error or it fails and leaves the database unchanged

- No explicit Begin_Transaction statement
  - Transaction initiation implicit at first SQL statement and at next SQL statement after previous transaction terminates

- Every transaction must have an explicit end statement
  - **COMMIT**: the DBMS must assure that the effects are permanent
  - **ROLLBACK**: the DBMS must assure that the effects are as if the transaction had not yet begun

LECTURE SUMMARY

- Transaction concepts
- ACID properties for transactions
- Transaction support in SQL